

# COMP9020 19T1

## Week 2

### Logic, Proofs, Boolean Algebra

- Textbook (R & W) - Ch. 2, Sec. 2.2-2.5  
Ch. 10, Sec. 10.1-10.5
- Problem set 2
- Supplementary Exercises Ch. 2, Ch. 10 (R & W)
- *Guidelines for good mathematical writing*
- Quiz 2 — due Monday 4 March (week 3) at 11am

# Logical Equivalence

Two formulas  $\phi, \psi$  are **logically equivalent**, denoted  $\phi \equiv \psi$  if they have the same truth value for all values of their basic propositions.

*Application:* If  $\phi$  and  $\psi$  are two formulae such that  $\phi \equiv \psi$ , then the digital circuits corresponding to  $\phi$  and  $\psi$  compute the same function. Thus, proving equivalence of formulas can be used to *optimise* circuits.

# Some Well-Known Equivalences

Excluded Middle  $p \vee \neg p \equiv \top$

Contradiction  $p \wedge \neg p \equiv \perp$

Identity  $p \vee \perp \equiv p$

$p \wedge \top \equiv p$

$p \vee \top \equiv \top$

$p \wedge \perp \equiv \perp$

Idempotence  $p \vee p \equiv p$

$p \wedge p \equiv p$

Double Negation  $\neg\neg p \equiv p$

Commutativity

$$p \vee q \equiv q \vee p$$

$$p \wedge q \equiv q \wedge p$$

Associativity

$$(p \vee q) \vee r \equiv p \vee (q \vee r)$$

$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$$

Distribution

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

De Morgan's laws

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

Implication

$$p \Rightarrow q \equiv \neg p \vee q$$

$$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$$

## Example

$$((r \wedge \neg p) \vee (r \wedge q)) \vee ((\neg r \wedge \neg p) \vee (\neg r \wedge q))$$

$$\equiv (r \wedge (\neg p \vee q)) \vee (\neg r \wedge (\neg p \vee q))$$

$$\equiv (r \vee \neg r) \wedge (\neg p \vee q)$$

$$\equiv \top \wedge (\neg p \vee q)$$

$$\equiv \neg p \vee q$$

Distrib.

Distrib.

Excl. Mid.

Ident.

## Exercise

2.2.18 Prove or disprove:

(a)  $p \Rightarrow (q \Rightarrow r) \equiv (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$

(c)  $(p \Rightarrow q) \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$

## Exercise

2.2.18 Prove or disprove:

$$\begin{aligned} \text{(a)} \quad & (p \Rightarrow q) \Rightarrow (p \Rightarrow r) \\ & \equiv \neg(p \Rightarrow q) \vee (\neg p \vee r) \\ & \equiv (p \wedge \neg q) \vee \neg p \vee r \\ & \equiv (p \vee \neg p \vee r) \wedge (\neg q \vee \neg p \vee r) \\ & \equiv \top \wedge (\neg p \vee \neg q \vee r) \\ & \equiv p \Rightarrow (\neg q \vee r) \\ & \equiv p \Rightarrow (q \Rightarrow r) \end{aligned}$$

$$\text{(c)} \quad (p \Rightarrow q) \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$$

Counterexample:

$p$	$q$	$r$	$(p \Rightarrow q) \Rightarrow r$	$p \Rightarrow (q \Rightarrow r)$
F	T	F	F	T

# Satisfiability of Formulas

A formula is **satisfiable**, if it evaluates to T for *some* assignment of truth values to its basic propositions.

## Example

$A$	$B$	$\neg(A \Rightarrow B)$
F	F	F
F	T	F
T	F	T
T	T	F



## Applications II: Constraint Satisfaction Problems

These are problems such as timetabling, activity planning, etc. Many can be understood as showing that a formula is satisfiable.

### Example

You are planning a party, but your friends are a bit touchy about who will be there.

- ❶ If John comes, he will get very hostile if Sarah is there.
- ❷ Sarah will only come if Kim will be there also.
- ❸ Kim says she will not come unless John does.

Who can you invite without making someone unhappy?

Translation to logic: let  $J, S, K$  represent “John (Sarah, Kim) comes to the party”. Then the constraints are:

①  $J \Rightarrow \neg S$

②  $S \Rightarrow K$

③  $K \Rightarrow J$

Thus, for a successful party to be possible, we want the formula  $\phi = (J \Rightarrow \neg S) \wedge (S \Rightarrow K) \wedge (K \Rightarrow J)$  to be satisfiable.

Truth values for  $J, S, K$  making this true are called *satisfying assignments*, or *models*.

We figure out where the conjuncts are false, below. (so blank = T)

$J$	$K$	$S$	$J \Rightarrow \neg S$	$S \Rightarrow K$	$K \Rightarrow J$	$\phi$
F	F	F				
F	F	T		F		F
F	T	F			F	F
F	T	T			F	F
T	F	F				
T	F	T	F	F		F
T	T	F				
T	T	T	F			F

Conclusion: a party satisfying the constraints can be held. Invite nobody, or invite John only, or invite Kim and John.

## Exercise

2.7.14 (supp)

Which of the following formulae are *always* true?

(a)  $(p \wedge (p \Rightarrow q)) \Rightarrow q$  — always true

(b)  $((p \vee q) \wedge \neg p) \Rightarrow \neg q$  — not always true

(e)  $((p \Rightarrow q) \vee (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$  — not always true

(f)  $(p \wedge q) \Rightarrow q$  — always true

## Exercise

2.7.14 (supp)

Which of the following formulae are *always* true?

(a)  $(p \wedge (p \Rightarrow q)) \Rightarrow q$  — always true

(b)  $((p \vee q) \wedge \neg p) \Rightarrow \neg q$  — not always true

(e)  $((p \Rightarrow q) \vee (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$  — not always true

(f)  $(p \wedge q) \Rightarrow q$  — always true

# Validity, Entailment, Arguments

An *argument* consists of a set of declarative sentences called *premises* and a declarative sentence called the *conclusion*.

## Example

Premises:	Frank took the Ford or the Toyota. If Frank took the Ford he will be late. Frank is not late.
Conclusion:	Frank took the Toyota

An argument is *valid* if the conclusions are true *whenever* all the premises are true. Thus: if we believe the premises, we should also believe the conclusion.

(Note: we don't care what happens when one of the premises is false.)

Other ways of saying the same thing:

- The conclusion *logically follows* from the premises.
- The conclusion is a *logical consequence* of the premises.
- The premises **entail** the conclusion.

The argument above is valid. The following is invalid:

### Example

Premises:     Frank took the Ford or the Toyota.  
                  If Frank took the Ford he will be late.  
                  Frank is late.

---

Conclusion:   Frank took the Ford.



For arguments in propositional logic, we can capture validity as follows:

Let  $\phi_1, \dots, \phi_n$  and  $\phi$  be formulae of propositional logic. Draw a truth table with columns for each of  $\phi_1, \dots, \phi_n$  and  $\phi$ .

The argument with premises  $\phi_1, \dots, \phi_n$  and conclusion  $\phi$  is valid, denoted

$$\phi_1, \dots, \phi_n \models \phi$$

if in every row of the truth table where  $\phi_1, \dots, \phi_n$  are all true,  $\phi$  is true also.

We mark only true locations (blank = F)

$Frd$	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \Rightarrow Late$	$\neg Late$	$Tyta$
F	F	F		T	T	
F	F	T		T		
F	T	F	T	T	T	T
F	T	T	T	T		T
T	F	F	T		T	
T	F	T	T	T		
T	T	F	T		T	T
T	T	T	T	T		T

This shows  $Frd \vee Tyta, Frd \Rightarrow Late, \neg Late \models Tyta$

The following row shows  $Frd \vee Tyta$ ,  $Frd \Rightarrow Late$ ,  $Late \not\Rightarrow Frd$

$Frd$	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \Rightarrow Late$	$Late$	$Frd$
F	T	T	T	T	T	F

## Applications III: Reasoning About Requirements/Specifications

Suppose a set of English language requirements  $R$  for a software/hardware system can be formalised by a set of formulae  $\{\phi_1, \dots, \phi_n\}$ .

Suppose  $C$  is a statement formalised by a formula  $\psi$ . Then

- 1 The requirements cannot be implemented if  $\phi_1 \wedge \dots \wedge \phi_n$  is not satisfiable.
- 2 If  $\phi_1, \dots, \phi_n \models \psi$  then every correct implementation of the requirements  $R$  will be such that  $C$  is always true in the resulting system.
- 3 If  $\phi_1, \dots, \phi_{n-1} \models \phi_n$ , then the condition  $\phi_n$  of the specification is redundant and need not be stated in the specification.

## Example

*Requirements R*: A burglar alarm system for a house is to operate as follows. The alarm should not sound unless the system has been armed or there is a fire. If the system has been armed and a door is disturbed, the alarm should ring. Irrespective of whether the system has been armed, the alarm should go off when there is a fire.

*Conclusion C*: If the alarm is ringing and there is no fire, then the system must have been armed.

### Question

- 1 *Will every system correctly implementing requirements R satisfy C?*
- 2 *Is the final sentence of the requirements redundant?*

Expressing the requirements as formulas of propositional logic,  
with

- $S$  = the alarm sounds = the alarm rings
- $A$  = the system is armed
- $D$  = a door is disturbed
- $F$  = there is a fire

we get

**Requirements:**

- 1  $S \Rightarrow (A \vee F)$
- 2  $(A \wedge D) \Rightarrow S$
- 3  $F \Rightarrow S$

**Conclusion:**  $(S \wedge \neg F) \Rightarrow A$

Our two questions then correspond to

- 1 Does  $S \Rightarrow (A \vee F), (A \wedge D) \Rightarrow S, F \Rightarrow S \models (S \wedge \neg F) \Rightarrow A$  ?
- 2 Does  $S \Rightarrow (A \vee F), (A \wedge D) \Rightarrow S \models F \Rightarrow S$  ?

Answers: problem set 2, exercise 2

# Validity of Formulas

A formula  $\phi$  is **valid**, or a **tautology**, denoted  $\models \phi$ , if it evaluates to T for *all* assignments of truth values to its basic propositions.

## Example

A	B	$(A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A)$
F	F	T
F	T	T
T	F	T
T	T	T



# Validity, Equivalence and Entailment

## Theorem

*The following are equivalent:*

- $\phi_1, \dots, \phi_n \models \psi$
- $\models (\phi_1 \wedge \dots \wedge \phi_n) \Rightarrow \psi$
- $\models \phi_1 \Rightarrow (\phi_2 \Rightarrow \dots (\phi_n \Rightarrow \psi) \dots)$

## Theorem

$\phi \equiv \psi$  if and only if  $\models \phi \Leftrightarrow \psi$

# Proof Rules and Methods:

## Proof by Cases

We want to prove that  $A$ . To prove it, we find a set of cases  $B_1, B_2, \dots, B_n$  such that

- 1  $B_1 \vee \dots \vee B_n$ , and
- 2  $B_i \Rightarrow A$  for each  $i = 1..n$ .

(Hard Part: working out what the  $B_i$  should be.)

(Comment: often  $n = 2$  and  $B_2 = \neg B_1$ , so  $B_1 \vee B_2 = B_1 \vee \neg B_1$  holds trivially.)

### Example

$|x + y| \leq |x| + |y|$  for all  $x, y \in \mathbb{R}$ .

Recall:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

# Quantifiers

We've made quite a few statements of the kind

*"If there exists a satisfying assignment . . ."*

or

*"Every natural number greater than 2 . . ."*

without formally capturing these quantitative aspects.

**Notation:**  $\forall$  means "for all" and  $\exists$  means "there exist(s)"

## Example

Goldbach's conjecture

$$\forall n \in 2\mathbb{N} (n > 2 \Rightarrow \exists p, q \in \mathbb{N} (p, q \in \text{PRIMES} \wedge n = p + q))$$

## Exercise

Which of the following is a tautology?

- $\forall x (\exists y (P(x, y))) \Rightarrow \exists y (\forall x (P(x, y)))$  not always true
- $\exists y (\forall x (P(x, y))) \Rightarrow \forall x (\exists y (P(x, y)))$  always true

## Exercise

Which of the following is a tautology?

- $\forall x (\exists y (P(x, y))) \Rightarrow \exists y (\forall x (P(x, y)))$  not always true
- $\exists y (\forall x (P(x, y))) \Rightarrow \forall x (\exists y (P(x, y)))$  always true

# Proof Rules and Methods:

## Proof of the Contrapositive

We want to prove  $A \Rightarrow B$ .

To prove it, we show  $\neg B \Rightarrow \neg A$  and invoke the equivalence  $(A \Rightarrow B) \equiv (\neg B \Rightarrow \neg A)$ .

### Example

$$\forall m, n \in \mathbb{N} (m + n \geq 73 \Rightarrow m \geq 37 \vee n \geq 37)$$

# Proof Rules and Methods:

## Proof by Contradiction

We want to prove  $A$ .

To prove it, we assume  $\neg A$ , and derive both  $B$  and  $\neg B$  for some proposition  $B$ .

(Hard part: working out what  $B$  should be.)

### Examples

- $\sqrt{2}$  is irrational
- There exist an infinite number of primes

## Substitution

*Substitution* is the process of replacing every occurrence of some symbol by an expression.

### Examples

The result of substituting 3 for  $x$  in

$$x^2 + 7y = 2xz$$

is

$$3^2 + 7y = 2 \cdot 3 \cdot z$$

The result of substituting  $2k + 3$  for  $x$  in

$$x^2 + 7y = 2xz$$

is

$$(2k + 3)^2 + 7y = 2 \cdot (2k + 3) \cdot z$$



We can substitute logical expressions for logical variables:

### Example

The result of substituting  $P \wedge Q$  for  $A$  in

$$(A \wedge B) \Rightarrow A$$

is

$$((P \wedge Q) \wedge B) \Rightarrow (P \wedge Q)$$

# Substitution Rules

(a) If we substitute an expression for *all* occurrences of a logical variable in a tautology then the result is still a tautology.

If  $\models \phi(P)$  then  $\models \phi(\alpha)$ .

## Examples

$\models P \Rightarrow (P \vee Q)$ , so

$$\models (A \vee B) \Rightarrow ((A \vee B) \vee Q)$$

2.5.7

$\models \neg Q \Rightarrow (Q \Rightarrow P)$ , so

$$\models \neg(P \Rightarrow Q) \Rightarrow ((P \Rightarrow Q) \Rightarrow P)$$

(b) If a logical formula  $\phi$  contains a formula  $\alpha$ , and we replace (an occurrence of)  $\alpha$  by a logically equivalent formula  $\beta$ , then the result is logically equivalent to  $\phi$ .

If  $\alpha \equiv \beta$  then  $\phi(\alpha) \equiv \phi(\beta)$ .

### Example

$P \Rightarrow Q \equiv \neg P \vee Q$ , so

$$Q \Rightarrow (P \Rightarrow Q) \equiv Q \Rightarrow (\neg P \vee Q)$$

# Boolean Functions

Formulae can be viewed as **Boolean functions** mapping valuations of their propositional letters to truth values.

A Boolean function of one variable is also called **unary**.

A function of two variables is called **binary**.

A function of  $n$  input variables is called **n-ary**.

## Question

*How many unary Boolean functions are there?*

*How many binary functions? n-ary?*

## Question

*What connectives do we need to express all of them?*

# Boolean Arithmetic

Consider truth values with operations  $\wedge, \vee, \neg$  as an **algebraic structure**:

- $\mathbb{B} = \{0, 1\}$  with 'Boolean' arithmetic

$$a \cdot b, a + b, a' = 1 - a$$

## NB

We often write  $pq$  for  $p \cdot q$ .

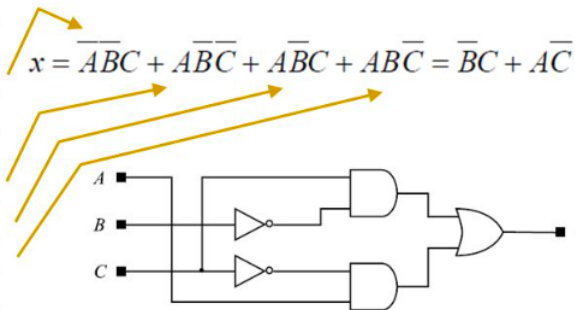
In electrical and computer engineering, the notation  $\overline{p}$  is more common than  $p'$ , which is often used in mathematics.

Observe that using  $\overline{(\cdot)}$  obviates the need for some parentheses.

## Applications IV: Digital Circuits

A formula can be viewed as defining a digital circuit, which computes a Boolean function of the input propositions. The function is given by the truth table of the formula.

$A$	$B$	$C$	$x$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



## Definition: Boolean Algebra

Every structure consisting of a set  $T$  with operations *join*:  $a, b \mapsto a + b$ , *meet*:  $a, b \mapsto a \cdot b$  and *complementation*:  $a \mapsto a'$ , and distinct elements 0 and 1, is called a **Boolean algebra** if it satisfies the following laws, for all  $x, y, z \in T$ :

**commutative:**     •  $x + y = y + x$

•  $x \cdot y = y \cdot x$

**associative:**     •  $(x + y) + z = x + (y + z)$

•  $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

**distributive:**     •  $x + (y \cdot z) = (x + y) \cdot (x + z)$

•  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

**identity:**  $x + 0 = x, \quad x \cdot 1 = x$

**complementation:**  $x + x' = 1, \quad x \cdot x' = 0$

## Exercise

**Example 10.1.2** Define a Boolean algebra for 2-bit vectors  $\mathbb{B}^2$

$$0 \stackrel{\text{def}}{=} (0, 0)$$

$$1 \stackrel{\text{def}}{=} (1, 1)$$

$$\text{join: } (a_1, a_2) + (b_1, b_2) \mapsto (a_1 + b_1, a_2 + b_2)$$

$$\text{meet: } (a_1, a_2) \cdot (b_1, b_2) \mapsto (a_1 \cdot b_1, a_2 \cdot b_2)$$

$$\text{complementation: } (a_1, a_2)' \mapsto (a_1', a_2')$$

Check that all Boolean algebra laws hold for  $x, y \in \mathbb{B} \times \mathbb{B}$



## Exercise

Example 10.1.2 Define a Boolean algebra for 2-bit vectors  $\mathbb{B}^2$

$$0 \stackrel{\text{def}}{=} (0, 0)$$

$$1 \stackrel{\text{def}}{=} (1, 1)$$

$$\text{join: } (a_1, a_2) + (b_1, b_2) \mapsto (a_1 + b_1, a_2 + b_2)$$

$$\text{meet: } (a_1, a_2) \cdot (b_1, b_2) \mapsto (a_1 \cdot b_1, a_2 \cdot b_2)$$

$$\text{complementation: } (a_1, a_2)' \mapsto (a_1', a_2')$$

Check that all Boolean algebra laws hold for  $x, y \in \mathbb{B} \times \mathbb{B}$

# Boolean Expressions

Boolean algebra (BA) notation for propositional formulae:

	PL	BA
propositional atoms	$p, q, \dots$	$p, q, \dots$
conjunction	$p \wedge q$	$p \cdot q$ or $pq$
disjunction	$p \vee q$	$p + q$
negation	$\neg p$	$p'$

## Example

$$(p \vee q) \wedge (\neg(p \vee \neg q) \vee \neg(\neg(r \wedge (p \vee \neg q))))$$

$$\begin{aligned} & (p + q) \cdot ((p + q')' + (r \cdot (p + q'))'') \\ &= (p + q)((p + q')' + (r(p + q'))'') \end{aligned}$$

# Terminology and Rules

- A **literal** is an expression  $p$  or  $p'$ , where  $p$  is a propositional atom.
- An expression is in CNF (conjunctive normal form) if it has the form

$$\prod_i C_i$$

where each **clause**  $C_i$  is a disjunction of literals e.g.  $p + q + r'$ .

- An expression is in DNF (disjunctive normal form) if it has the form

$$\sum_i C_i$$

where each clause  $C_i$  is a conjunction of literals e.g.  $pqr'$ .

- CNF and DNF are named after their top level operators; no deeper nesting of  $\cdot$  or  $+$  is permitted.
- We can assume in every clause (disjunct for the CNF, conjunct for the DNF) any given variable (literal) appears only once; preferably, no literal and its negation together.
  - $x + x = x$ ,  $xx = x$
  - $xx' = 0$ ,  $x + x' = 1$
  - $x \cdot 0 = 0$ ,  $x \cdot 1 = x$ ,  $x + 0 = x$ ,  $x + 1 = 1$
- A preferred form for an expression is DNF, with as few terms as possible. In deriving such minimal simplifications the two basic rules are **absorption** and **combining the opposites**.

### Fact

- 1  $x + xy = x$  (*absorption*)
- 2  $xy + xy' = x$  (*combining the opposites*)

## Theorem

*For every Boolean expression  $\phi$ , there exists an equivalent expression in conjunctive normal form and an equivalent expression in disjunctive normal form.*

## Proof.

We show how to apply the equivalences already introduced to convert any given formula to an equivalent one in CNF, DNF is similar. □

## Step 1: Push Negations Down

Using **De Morgan's** laws and the **double negation** rule

$$(x + y)' = x' \cdot y'$$

$$(x \cdot y)' = x' + y'$$

$$(x')' = x$$

we push negations down towards the atoms until we obtain a formula that is formed from literals using only  $\cdot$  and  $+$ .

## Step 2: Use Distribution to Convert to CNF

Using the distribution rules

$$x + (y_1 \cdot \dots \cdot y_n) = (x + y_1) \cdot \dots \cdot (x + y_n)$$

$$(y_1 \cdot \dots \cdot y_n) + x = (y_1 + x) \cdot \dots \cdot (y_n + x)$$

we obtain a CNF formula.

# CNF/DNF in Propositional Logic

Using the equivalence

$$A \Rightarrow B \equiv \neg A \vee B$$

we first eliminate all occurrences of  $\Rightarrow$

## Example

$$\neg(\neg p \wedge ((r \wedge s) \Rightarrow q)) \equiv \neg(\neg p \wedge (\neg(r \wedge s) \vee q))$$



Step 1:

### Example

$$\begin{aligned}(p'((rs)' + q))' &= (p')' + ((rs)' + q)' \\ &= p + (rs')' \cdot q' \\ &= p + rsq'\end{aligned}$$

Step 2:

### Example

$$\begin{aligned}p + rsq' &= (p + r)(p + sq') \\ &= (p + r)(p + s)(p + q') \quad \text{CNF}\end{aligned}$$

## Canonical Form DNF

Given a Boolean expression  $E$ , we can construct an equivalent DNF  $E^{dnf}$  from the lines of the truth table where  $E$  is true:

Given an assignment  $\pi$  of 0, 1 to variables  $x_1 \dots x_i$ , define the literal

$$\ell_i = \begin{cases} x_i & \text{if } \pi(x_i) = 1 \\ x_i' & \text{if } \pi(x_i) = 0 \end{cases}$$

and a product  $t_\pi = \ell_1 \cdot \ell_2 \cdot \dots \cdot \ell_n$ .

### Example

If  $\pi(x_1) = 1$  and  $\pi(x_2) = 0$  then  $t_\pi = x_1 \cdot x_2'$

The **canonical DNF** of  $E$  is

$$E^{dnf} = \sum_{E(\pi)=1} t_\pi$$

## Example

If  $E$  is defined by

$x$	$y$	$E$
0	0	1
0	1	0
1	0	1
1	1	1

then  $E^{dnf} = x'y' + xy' + xy$

Note that this can be simplified to either

$$y' + xy$$

or

$$x'y' + x$$

## Exercise

10.2.3 Find the canonical DNF form of each of the following expressions in variables  $x, y, z$

- $xy$
- $z'$
- $xy + z'$
- $1$

## Exercise

**10.2.3** Find the canonical DNF form of the following expressions in the three variables  $x, y, z$ .

$$xy = xy \cdot 1 = xy \cdot (z + z') = xyz + xyz'$$

$$z' = xyz' + xy'z' + x'yz' + x'y'z'$$

$xy + z' =$  combine all of the 5 different product terms above

$1 =$  sum of all 8 possible product terms:

$$xyz + x'yz + \dots + x'y'z'$$

## NB

Obviously, preferred in practice are the expressions with as few terms as possible.

However, the existence of a uniform representation as the sum of (quite a few) product terms is important for proving the properties of Boolean expressions.

# Karnaugh Maps

For up to four variables (propositional symbols) a diagrammatic method of simplification called **Karnaugh maps** works quite well. For every propositional function of  $k = 2, 3, 4$  variables we construct a rectangular array of  $2^k$  cells. We mark the squares corresponding to the value 1 with eg “+” and try to cover these squares with as few rectangles with sides 1 or 2 or 4 as possible.

## Example

10.4.2 Use a K-map to find an optimised form.

	$yz$	$yz'$	$y'z'$	$y'z$
$x$	+	+		+
$x'$	+		+	+

For optimisation, the idea is to cover the + squares with the minimum number of rectangles. One *cannot* cover any empty cells (they indicate where  $f(w, x, y, z)$  is 0).

- The rectangles can go 'around the corner'/the actual map should be seen as a *torus*.
- Rectangles must have sides of 1, 2 or 4 squares (three adjacent cells are useless).

## Exercise

	$yz$	$yz'$	$y'z'$	$y'z$
$x$	+	+		+
$x'$	+		+	+







$$f = xy + x'y' + z$$

Canonical form would consist of writing all cells separately:  
 $xyz + xyz' + xy'z + x'yz + x'y'z' + x'y'z$

For optimisation, the idea is to cover the + squares with the minimum number of rectangles. One *cannot* cover any empty cells (they indicate where  $f(w, x, y, z)$  is 0).

- The rectangles can go 'around the corner'/the actual map should be seen as a *torus*.
- Rectangles must have sides of 1, 2 or 4 squares (three adjacent cells are useless).

## Exercise

	$yz$	$yz'$	$y'z'$	$y'z$
$x$				
$x'$				

$$f = xy + x'y' + z$$

Canonical form would consist of writing all cells separately:

$$xyz + xyz' + xy'z + x'yz + x'y'z' + x'y'z$$



## Supplementary Exercise

### Exercise

10.6.6(c)

	$yz$	$yz'$	$y'z'$	$y'z$
$wx$	+	+		+
$wx'$	+	+	+	+
$w'x'$			+	+
$w'x$	+			+

$$f = wy + x'y' + xz$$

Note: trying to use  $wx'$  or  $y'z$  doesn't give as good a solution

## Supplementary Exercise

### Exercise

10.6.6(c)

	$yz$	$yz'$	$y'z'$	$y'z$
$wx$	+	+		+
$wx'$	+	+	+	+
$w'x'$			+	+
$w'x$	+			+

$$f = wy + x'y' + xz$$

Note: trying to use  $wx'$  or  $y'z$  doesn't give as good a solution

# Boolean Algebras in Computer Science

Several data structures have natural operations following essentially the same rules as logical  $\wedge$ ,  $\vee$  and  $\neg$ .

- $n$ -tuples of 0's and 1's with Boolean operations, e.g.

$$\textit{join: } (1, 0, 0, 1) + (1, 1, 0, 0) = (1, 1, 0, 1)$$

$$\textit{meet: } (1, 0, 0, 1) \cdot (1, 1, 0, 0) = (1, 0, 0, 0)$$

$$\textit{complementation: } (1, 0, 0, 1)' = (0, 1, 1, 0)$$

- $\text{Pow}(S)$  — subsets of  $S$

$$\textit{join: } A \cup B, \quad \textit{meet: } A \cap B, \quad \textit{complement: } A^c = S \setminus A$$

# Example

## Exercise

**Example 10.1.1** Define a Boolean algebra for the power set  $\text{Pow}(S)$  of  $S = \{a, b, c\}$

$$0 \stackrel{\text{def}}{=} \emptyset$$

$$1 \stackrel{\text{def}}{=} \{a, b, c\}$$

$$\text{join: } X, Y \mapsto X \cup Y$$

$$\text{meet: } X, Y \mapsto X \cap Y$$

$$\text{complementation: } X \mapsto \{a, b, c\} \setminus X$$

Additional exercise:

Verify that all Boolean algebra laws (cf. slide 39) hold for  $X, Y, Z \in \text{Pow}(\{a, b, c\})$

# Example

## Exercise

**Example 10.1.1** Define a Boolean algebra for the power set  $\text{Pow}(S)$  of  $S = \{a, b, c\}$

$$0 \stackrel{\text{def}}{=} \emptyset$$

$$1 \stackrel{\text{def}}{=} \{a, b, c\}$$

$$\text{join: } X, Y \mapsto X \cup Y$$

$$\text{meet: } X, Y \mapsto X \cap Y$$

$$\text{complementation: } X \mapsto \{a, b, c\} \setminus X$$

Additional exercise:

Verify that all Boolean algebra laws (cf. slide 39) hold for  $X, Y, Z \in \text{Pow}(\{a, b, c\})$

# More Examples of Boolean Algebras in CS

- Functions from any set  $S$  to  $\mathbb{B}$ ; their set is denoted  $\text{Map}(S, \mathbb{B})$

If  $f, g : S \longrightarrow \mathbb{B}$  then

- $(f + g) : S \longrightarrow \mathbb{B}$  is defined by  $s \mapsto f(s) + g(s)$
- $(f \cdot g) : S \longrightarrow \mathbb{B}$  is defined by  $s \mapsto f(s) \cdot g(s)$
- $f' : S \longrightarrow \mathbb{B}$  is defined by  $s \mapsto (f(s))'$

There are  $2^n$  such functions for  $|S| = n$

- All Boolean functions of  $n$  variables, e.g.

$$(p_1, p_2, p_3) \mapsto (p_1 + p_2') \cdot (p_1 + p_3) \cdot p_2 + p_3'$$

There are  $2^{2^n}$  of them; their collection is denoted  $\text{BOOL}(n)$

## Fact

Every finite Boolean algebra satisfies:  $|T| = 2^k$  for some  $k$ .

## Definition

Consider

- Boolean algebra  $B_1$  over a set  $S$  with distinct elements  $0_S, 1_S$
- Boolean algebra  $B_2$  over a set  $T$  with distinct elements  $0_T, 1_T$

They are **isomorphic**, written  $B_1 \simeq B_2$ , if and only if there is a one-to-one correspondence  $\iota : S \mapsto T$  such that

- 1  $\iota(0_S) = 0_T$
- 2  $\iota(1_S) = 1_T$
- 3  $\iota(s_1 + s_2) = \iota(s_1) + \iota(s_2)$  for all  $s_1, s_2 \in S$
- 4  $\iota(s_1 \cdot s_2) = \iota(s_1) \cdot \iota(s_2)$  for all  $s_1, s_2 \in S$
- 5  $\iota(s') = \iota(s)'$  for all  $s \in S$

## Fact

*All algebras with the same number of elements are **isomorphic**, i.e. “structurally similar”. Therefore, studying one such algebra describes properties of all.*

A cartesian product of Boolean algebras is again a Boolean algebra. We write

$$\mathbb{B}^k = \mathbb{B} \times \dots \times \mathbb{B}$$

The algebras mentioned above are all of this form

- $n$ -tuples  $\simeq \mathbb{B}^n$
- $\text{Pow}(S) \simeq \mathbb{B}^{|S|}$
- $\text{Map}(S, \mathbb{B}) \simeq \mathbb{B}^{|S|}$
- $\text{BOOL}(n) \simeq \mathbb{B}^{2^n}$

## NB

Boolean algebra as the calculus of two values is fundamental to computer circuits and computer programming.

Example: Encoding subsets as bit vectors.



# Summary

- equivalence  $\equiv$  , some well-known equivalences (slides 3–4)
- satisfiable formulae, valid formulae (tautologies)
- logical entailment  $\models$
- Proof methods: contrapositive, by contradiction, by cases
- Boolean algebra, CNF, DNF, canonical form

Supplementary reading [LLM]

- Ch. 1, Sec. 1.5-1.9 (more about good proofs)
- Ch. 3, Sec. 3.3 (more about proving equivalences of formulae)